Evolutionary Algorithms: GA & GP

Jiří Kubalík
Department of Cybernetics, CTU Prague
Contents

- Genetic Algorithms (GAs)
  - Simple Genetic Algorithm (SGA)
  - Areas for EA’s applications
  - SGA example: Evolving strategy for an artificial ant problem
  - Schema theory – a schema, its properties, exponential growth equation and its consequences
- Genetic Programming (GP)
  - Tree representation, closure condition, 'strong typing'
  - Application of GP to artificial ant problem
  - Other examples
Evolutionary Algorithms: Characteristics

EA are stochastic optimization algorithms

- **Stochastic** – but not random search,

- **Use an analogy of natural evolution**
  - genetic inheritance (J.G. Mendel) – the basic principles of transference of hereditary factors from parent to offspring – genes, which present hereditary factors, are lined up on chromosomes.
  - strife for survival (Ch. Darwin) – the fundamental principle of natural selection – is the process by which individual organisms with favorable traits are more likely to survive and reproduce.

- **Not fast in some sense** – population-based algorithm,

- **Robust** – efficient in finding good solutions in difficult searches.
EA: Vocabulary

Vocabulary borrowed from natural genetics

- **Individual** (chromosome + its quality measure ”fitness value”) – a solution to a problem.
- **Chromosome** – entire representation of the solution.
- **Fitness** – quality measure assigned to an individual, expresses how well it is adapted to the environment.
- **Gene** (also features, characters) – elementary units from which chromosomes are made.
  - each gene is located at certain place of the chromosome called locus (pl. loci),
  - a particular value for a locus is an allele.
    example: the ”thickness” gene (which might be at locus 8) might be set to allele 2, meaning its second-thinnest value.
- **Genotype** – what’s on the chromosome.
- **Phenotype** – what it means in the problem context (e.g., binary sequence may map to integers or reals, or order of execution, etc.).
Problem can be represented as

- **binary string** – 1 0 1 1 0 1 1 0 0 1 0 1 1 0 1
- **real-valued string** – 3.24 1.78 -2.61
- **string of chars** – D → E → A → C → B
- **or as a tree**

![Tree Diagram](image)

- **or as a graph**, and others.
Evaluation Function

Objective (Fitness) function
- the only information about the sought solution the algorithm dispose of,
- must be defined for every possible chromosome.

Fitness function may be
- multimodal,
- discrete,
- multidimensional,
- nonlinear,
- noisy,
- multiobjective.

Fitness does not have to be define analytically
- simulation results,
- classification success rate.

Fitness function should not be too costly!!!
Example: Coding & Evaluation

Function optimization

- maximization of \( f(x, y) = x^2 + y^2 \),
- parameters \( x \) and \( y \) take on values from interval \( <0, 31> \),
- and are code on 5 bits each.

<table>
<thead>
<tr>
<th>genotype</th>
<th>phenotype</th>
<th>fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000, 01010</td>
<td>0, 10</td>
<td>100</td>
</tr>
<tr>
<td>00001, 11001</td>
<td>1, 25</td>
<td>625 + 1 = 626</td>
</tr>
<tr>
<td>01011, 00011</td>
<td>11, 3</td>
<td>121 + 9 = 130</td>
</tr>
<tr>
<td>11011, 10010</td>
<td>27, 18</td>
<td>729 + 324 = 1053</td>
</tr>
</tbody>
</table>
Evolutionary Cycle

- **Population**
  - Selection
  - Replacement

- **Parents**
  - Recombination
  - Mutation
  - Evaluation

- **Offspring**
Idealized Illustration of Evolution

- Uniformly sampled population.
- Population converged to promising regions.
Initialization

Random
- randomly generated solutions,
- no prior information about the shape of the sought solution,
- relies just on "lucky" sampling of the whole search space by a finite set of samples.

Informed (pre-processing)
- (meta)heuristic routines used for seeding the initial population,
- biased random generator sampling regions of the search space that are likely to contain the sought solutions,
  + may help to find better solutions,
  + may speed up the search process,
  - may cause irreversible focusing of the search process on regions with local optima.
Reproduction

Models nature’s survival-of-fittest principle
- prefers better individuals to the worse ones,
- still, every individual should have a chance to reproduce.

Roulette wheel
- probability of choosing some solution is directly proportional to its fitness value

\[ P_i = \frac{f_i}{\text{PopSize}} \sum_{j=1}^{\text{PopSize}} f_j \]

Other methods
- Stochastic Universal Sampling,
- Tournament selection,
- Reminder Stochastic Sampling.
Genetic Operators: Crossover

Idea
■ given two well-fit solutions to the given problem, it is possible to get a new solution by properly mixing the two that is even better than both its parents.

Role of crossover
■ sampling (exploration) of the search space

Example: 1-point crossover

![Diagram showing 1-point crossover](image)
Genetic Operators: Mutation

Role of mutation

- preservation of a population diversity,
- minimization of a possibility of losing some important piece of genetic information.

![Diagram illustrating single bit-flipping mutation]

<table>
<thead>
<tr>
<th>Population</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 1 1 0 0 1 1 0</td>
<td></td>
</tr>
<tr>
<td>0 1 1 0 0 1 0 1 0</td>
<td></td>
</tr>
<tr>
<td>0 0 0 1 1 0 1 0 1</td>
<td></td>
</tr>
<tr>
<td>0 1 0 0 1 0 0 1 1</td>
<td></td>
</tr>
<tr>
<td>0 1 1 0 0 0 0 1 0</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>0 1 0 0 1 1 0 1 0</td>
<td></td>
</tr>
<tr>
<td>0 1 0 0 1 1 0 1 0</td>
<td></td>
</tr>
</tbody>
</table>

Example of missing genetic information
Replacement Strategy

Replacement strategy defines

- how big portion of the current generation will be replaced in each generation, and
- which solutions in the current population will be replaced by the newly generated ones.

Two extreme cases

- **Generational** – the whole old population is completely rebuilt in each generation (analogy of short-lived species).
- **Steady-state** – just a few individuals are replaced in each generation (analogy of longer-lived species).
Application Areas of Evolutionary Algorithms

EAs are popular for their
- simplicity,
- effectiveness,
- robustness.

Holland: “It’s best used in areas where you don’t really have a good idea what the solution might be. And it often surprises you with what you come up with.”

Applications
- control,
- engineering design,
- image processing,
- planning & scheduling,
- VLSI circuit design,
- network optimization & routing problems,
- optimal resource allocation,
- marketing,
- credit scoring & risk assessment,
- and many others.
Multiple Traveling Salesmen Problem

Rescue operations planning

- Given $N$ cities and $K$ agents, find an optimal tour for each agent so that every city is visited exactly once.

- A typical criterion to be optimized is the overall time spent by the squad (i.e., the slowest team member) during the task execution.
Artificial Ant Problem

Santa Fe trail

- 32 × 32 grid with 89 food pieces.
- Obstacles
  - 1×, 2× strait,
  - 1×, 2×, 3× right/left.

Ant capabilities

- detects the food right in front of him in direction he faces.
- actions observable from outside
  - MOVE – makes a step and eats a food piece if there is some,
  - LEFT – turns left,
  - RIGHT – turns right,
  - NO-OP – no operation.

Goal is to find a strategy that would navigate an ant through the grid so that it finds all the food pieces in the given time (600 time steps).
Artificial Ant Problem: GA Approach

Collins a Jefferson 1991, standard GA using binary representation

Representation

- strategy represented by finite state machine,
- table of transitions coded as binary chromosomes of fixed length.

Example: 4-state FSM, 34-bit long chromosomes

<table>
<thead>
<tr>
<th>Current state</th>
<th>Input</th>
<th>New state</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>00</td>
<td>0</td>
<td>01</td>
</tr>
<tr>
<td>2</td>
<td>00</td>
<td>1</td>
<td>00</td>
</tr>
<tr>
<td>3</td>
<td>01</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>01</td>
<td>1</td>
<td>00</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>1</td>
<td>00</td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>8</td>
<td>11</td>
<td>1</td>
<td>00</td>
</tr>
</tbody>
</table>

| 00 | 0110 | 0011 | 1001 | 0011 | 1101 | 0011 | 0010 | 0011 |

GA & GP
Artificial Ant Problem: Example cont.

**Ant behavior**

- What happens if the ant hits an obstacle?
- What is strange with transition from state 10 to the initial state 00?
- When does the ant succeed?
- Is the number of states sufficient to solve the problem?
- Do all of the possible 32-bit chromosomes represent a feasible solution?
Artificial Ant Problem: GA result

Representation
- 32 states,
- $453 = 64 \times 7 + 5$ bits !!!

Population size: 65,536 !!!

Number of generations: 200

Total number of samples tried: $13 \times 10^6$ !!!
Genetic Programming (GP)

GP shares with GA the philosophy of survival and reproduction of the fittest and the analogy of naturally occurring genetic operators.

GP differs from GA in a representation, genetic operators and a scope of applications.

GP is extension of the conventional GA in which the structures undergoing adaptation are trees of dynamically varying size and shape representing hierarchical computer programs.

Applications
- learning programs,
- learning decision trees,
- learning rules,
- learning strategies,
- . . .
All possible trees are composed of **functions** (inner nodes) and **terminals** (leaf nodes) appropriate to the problem domain.

- **Terminals** – inputs to the programs (independent variables), real, integer or logical constants, actions.

- **Functions**
  - arithmetic operators (+, -, *, /),
  - algebraic functions (sin, cos, exp, log),
  - logical functions (AND, OR, NOT),
  - conditional operators (If-Then-Else, cond?true:false),
  - and others.

**Closure** – each of the functions should be able to accept, as its argument, any value that may be returned by any function and any terminal.

**Example:** Tree representation of a LISP S-expression 0.23 * Z + X − 0.78
GP Initialisation: Common Methods

GP needs a good tree-creation algorithm to create trees for the initial population and subtrees for subtree mutation.

Required characteristics:

- Light computationally complex; optimally linear in tree size.
- User control over expected tree size.
- User control over specific node appearance in trees.

GROW method (each branch has depth $\leq D$):
- nodes at depth $d < D_{\text{max}}$ randomly chosen from $F \cup T$,
- nodes at depth $d = D_{\text{max}}$ randomly chosen from $T$.

FULL method (each branch has depth $= D$):
- nodes at depth $d < D$ randomly chosen from function set $F$,
- nodes at depth $d = D$ randomly chosen from terminal set $T$.

```
GROW(depth d, max depth D)
Returns: a tree of depth $\leq D - d$
1 if (d = D) return a random terminal
2 else
3   choose a random func or term f
4   if (f is terminal) return f
5   else
6     for each argument a of f
7       fill a with GROW(d + 1, D)
8     return f
```
GP Initialisation

Characteristics of GROW:

- does not have a size parameter – does not allow the user to create a desired size distribution,
- does not allow the user to define the expected probabilities of certain nodes appearing in trees,
- does not give the user much control over the tree structures generated.
- there is no appropriate way to create trees with either a fixed or average tree size or depth.

RAMPED HALF-AND-HALF – GROW & FULL method each deliver half of the initial population. 
$D$ is chosen between 2 to 6,
**GP Initialisation**

Characteristics of GROW:

- does not have a size parameter – does not allow the user to create a desired size distribution,
- does not allow the user to define the expected probabilities of certain nodes appearing in trees,
- does not give the user much control over the tree structures generated.
- there is no appropriate way to create trees with either a fixed or average tree size or depth.

**RAMPED HALF-AND-HALF – GROW & FULL** method each deliver half of the initial population. 
$D$ is chosen between 2 to 6,

**PTC1** is a modification of GROW that

- allows the user to define probabilities of appearance of functions within the tree,
- gives user a control over desired expected tree size, and guarantees that, on average, trees will be of that size.
- does not give the user any control over the variance in tree sizes,
- is fast, running in time near-linear in tree size.
GP: Standard Crossover

Parent 1: $Z \times Y \times (Y + 0.31 \times Z)$

Parent 2: $0.23 \times Z + X - 0.78$

Child 1: $0.23 \times Y \times Z^2$

Child 2: $Y + 0.31 \times Z + X - 0.78$
**GP: Subtree-Replacing Mutation**

*Mutation* replaces selected subtree with a randomly generated new one.

![Diagram of mutation process]
GP: Selection

Commonly used are the fitness proportionate roulette wheel selection or the tournament selection. **Greedy over-selection** is recommended for complex problems that require large populations (> 1000) – the motivation is to increase efficiency by increasing the chance of being selected to the fitter individuals in the population.

- rank population by fitness and divide it into two groups:
  - group I: the fittest individuals that together accounting for $c = x\%$ of the sum of fitness values in the population,
  - group II: remaining less fit individuals.

- 80% of the time an individual is selected from group I in proportion to its fitness; 20% of the time, an individual is selected from group II.

- For population size = 1000, 2000, 4000, 8000, $x = 32\%, 16\%, 8\%, 4\%$. 
  
%’s come from rule of thumb.
**GP: Selection**

Commonly used are the fitness proportionate roulette wheel selection or the tournament selection. **Greedy over-selection** is recommended for complex problems that require large populations (> 1000) – the motivation is to increase efficiency by increasing the chance of being selected to the fitter individuals in the population.

- rank population by fitness and divide it into two groups:
  - group I: the fittest individuals that together accounting for $c = x\%$ of the sum of fitness values in the population,
  - group II: remaining less fit individuals.

- 80% of the time an individual is selected from group I in proportion to its fitness; 20% of the time, an individual is selected from group II.

- For population size = 1000, 2000, 4000, 8000, $x = 32\%, 16\%, 8\%, 4\%$.
  
  %’s come from rule of thumb.

Example: Effect of greedy over-selection for the 6-multiplexer problem

<table>
<thead>
<tr>
<th>Population size</th>
<th>$I(M,i,z)$ without over-selection</th>
<th>$I(M,i,z)$ with over-selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000</td>
<td>343,000</td>
<td>33,000</td>
</tr>
<tr>
<td>2,000</td>
<td>294,000</td>
<td>18,000</td>
</tr>
<tr>
<td>4,000</td>
<td>160,000</td>
<td>24,000</td>
</tr>
</tbody>
</table>
**Artificial Ant Problem**

**Santa Fe trail**

- **32 × 32 grid** with 89 food pieces.
- **Obstacles**
  - 1×, 2× strait,
  - 1×, 2×, 3× right/left.

**Ant capabilities**

- **detects** the food right in front of him in direction he faces.
- **actions** observable from outside
  - MOVE – makes a step and eats a food piece if there is some,
  - LEFT – turns left,
  - RIGHT – turns right,
  - NO-OP – no operation.

**Goal** is to find a strategy that would navigate an ant through the grid so that it finds all the food pieces in the given time (600 time steps).
Artificial Ant Problem: GP Approach

**Terminals**
- motorial section,
- \( T = \text{MOVE, LEFT, RIGHT}. \)

**Functions**
- conditional IF-FOOD-AHEAD – food detection, 2 arguments (is/is_not food ahead),
- unconditional PROG2, PROG3 – sequence of 2/3 actions.

Ant repeats the program until time runs out (600 time steps) or all the food has been eaten.
Typical solutions in the initial population

- this solution

```
PROG2
```

- similarly this one

```
(IF-FOOD-AHEAD (LEFT) (RIGHT))
```

- this one

```
(PROG2 (MOVE) (MOVE))
```

just by chance finds 3 pieces of food.
More interesting solutions

- **Quilter** – performs systematic exploration of the grid,
  
  \[
  (\text{PROG3} \ (\text{RIGHT}) \\
  \ (\text{PROG3} \ (\text{MOVE}) \ (\text{MOVE}) \ (\text{MOVE})) \\
  \ (\text{PROG2} \ (\text{LEFT}) \ (\text{MOVE})))
  \]

- **Tracker** – perfectly tracks the food until the first obstacle occurs, then it gets trapped in an infinite loop.
  
  \[
  (\text{IF-FOOD-AHEAD} \ (\text{MOVE}) \ (\text{RIGHT}))
  \]
Artificial Ant Problem: GP Approach cont.

- **Avoider** – perfectly avoids food!!!
  
  \[
  (\text{I-F-A} \quad (\text{RIGHT})
  
  (\text{I-F-A} \quad (\text{RIGHT})
  
  (\text{PROG2} \quad (\text{MOVE}) \quad (\text{LEFT})))
  \]

Average fitness in the initial population is 3.5
Artificial Ant Problem: GP result

In generation 21, the following solution was found that already navigates an ant so that he eats all 89 food pieces in the given time.

\[
(I-F-A \ (MOVE) \\
(\text{PROG3} \ (I-F-A \ (MOVE) \\
(\text{RIGHT}) \\
(\text{PROG2} \ (\text{RIGHT}) \\
(\text{PROG2} \ (\text{LEFT}) \\
(\text{RIGHT})))) \\
(\text{PROG2} \ (I-F-A \ (MOVE) \\
(\text{LEFT})) \\
(MOVE))))
\]

This program solves every trail with the obstacles of the same type as occurs in Santa Fe trail.

**Compare the computational complexity with the GA approach!!!**

GA approach: \(65.536 \times 200 = 13 \times 10^6\) trials.

vs.

GP approach: \(500 \times 21 = 10.500\) trials.
Example of GP in Action: Trigonometric Identity

**Task** is to find an equivalent expression to $\cos(2x)$.

**GP implementation:**
- **Terminal set** $T = \{x, 1.0\}$.
- **Function set** $F = \{+, -, \ast, \%$, $\sin\}$.
- **Training cases**: 20 pairs $(x_i, y_i)$, where $x_i$ are values evenly distributed in interval $(0, 2\pi)$.
- **Fitness**: Sum of absolute differences between desired $y_i$ and the values returned by generated expressions.
- **Stopping criterion**: A solution found that gives the error less than 0.01.
Example of GP in Action: Trigonometric Identity cont.

1. **run, 13th generation**

\[ (- (- 1 (* (\sin x) (\sin x))) (* (\sin x) (\sin x))) \]

which equals (after editing) to \( 1 - 2 \sin^2 x \).

2. **run, 34th generation**

\[ (- 1 (* (* (\sin x) (\sin x)) 2)) \]

which is just another way of writing the same expression.
Example of GP in Action: Trigonometric Identity cont.

1. run, $13^{th}$ generation

\[-(-1 \ast (\sin x)(\sin x)) \ast (\sin x)(\sin x)\]

which equals (after editing) to $1 - 2 \ast \sin^2 x$.

2. run, $34^{th}$ generation

\[-1 \ast (\ast (\sin x)(\sin x)2)\]

which is just another way of writing the same expression.

3. run, $30^{th}$ generation

\[(\sin (-(-2\ast x2))\]

\[(\sin (\sin (\sin (\sin (\sin (\sin (\sin (\sin 1))\]

\[\sin (\sin 1))\]

\[)))]])))])\]
Example of GP in Action: Trigonometric Identity cont.

1. run, 13\textsuperscript{th} generation

\[ (- (- 1 (* (\sin x) (\sin x))) (* (\sin x) (\sin x))) \]

which equals (after editing) to \( 1 - 2 \times \sin^2 x \).

2. run, 34\textsuperscript{th} generation

\[ (- 1 (* (* (\sin x) (\sin x)) 2)) \]

which is just another way of writing the same expression.

3. run, 30\textsuperscript{th} generation

\[ (\sin (- (- 2 (* x 2))) \]

\[ (\sin (\sin (\sin (\sin (\sin (* (\sin (\sin 1)) \]

\[ (\sin (\sin 1)) \]

\[ ))))))))) \]

(2 minus the expression on the 2nd and 3rd rows) is almost \( \pi/2 \) so the discovered identity is

\[ \cos(2x) = \sin(\pi/2 - 2x). \]
EA Materials: Reading, Demos, Software

Reading

HUMIES: Human-Competitive Results
■ http://www.genetic-programming.org/hc2011/combined.html

Demos

Software